Deploying LLM-Powered Applications

Malay Agarwal

Contents

Questions to Ask When Deploying LLMs

There are a number of important questions to ask when deploying LLMs.

How Will the Model Function in Deployment?

The first set of questions is related to how the LLM will function in deployment:

- How fast do we need our model to generate completions?

- What compute budget do we have available?
- Are we willing to trade off model performance for improved inference speed or lower storage?

Does the Model Need Additional Resources?

The next set of questions is related to additional resources that the model may need:

- Will the model interact with external data or other applications?
- If it needs to do this, how will we connect to those resources?

How Will the Model Be Consumed?

Finally, there are questions related to how the model will be consumed:

- What will the intended application or API interface that the model will be consumed through will look like?

Model Optimization by Reducing Model Size for Deployment

Introduction

LLMs present inference challenges in terms of computing and storage requirements, as well as in ensuring low latency for consuming applications. These challenges are present irrespective of whether we are deploying on premises or to the cloud, and are even more prevalent when deploying to edge devices.

Reducing the size of the LLM is one of the primary ways to improve application performance. Reducing the size allows for quicker loading of the model, which reduces inference latency.

However, the challenge is to reduce the size of the LLM while still maintaining model performance.

Techniques

The techniques available have a trade-off between accuracy and performance.

> **Note**: Not all techniques available for reducing model size in general work well with generative models.

Distillation

Distillation is a technique that involves using a larger LLM called the *teacher model* to train a smaller LLM called the *student model*.

The student model learns to statistically mimic the behavior of the teacher model, either just in the final prediction layer or in the model's hidden layers as well.

When training a student model to mimic the behavior of the teacher model just in the final prediction, we have the following steps:

- Start with our fine-tuned LLM as the teacher model and create a smaller LLM for the student model. The weights of the teacher model are frozen.
- Use the teacher model to generate completions for the training data. At the same time, generate completions for the training data using the student model
- "Distill" the knowledge from the teacher model to the student model by minimizing a loss function which is a combination of a loss called the *student loss* and a loss called the *distillation loss*, given by:

$$\mathcal{L}(x;W) = \alpha * \mathcal{H}(y, \sigma(z_s; T=1)) + \beta * \mathcal{H}(\sigma(z_t; T=\tau), \sigma(z_s, T=\tau))$$

Here, $\mathcal{H}(y, \sigma(z_s; T=1))$ is the student loss and $\mathcal{H}(\sigma(z_t; T=\tau), \sigma(z_s, T=\tau))$ is the distillation loss.

In the above equation:

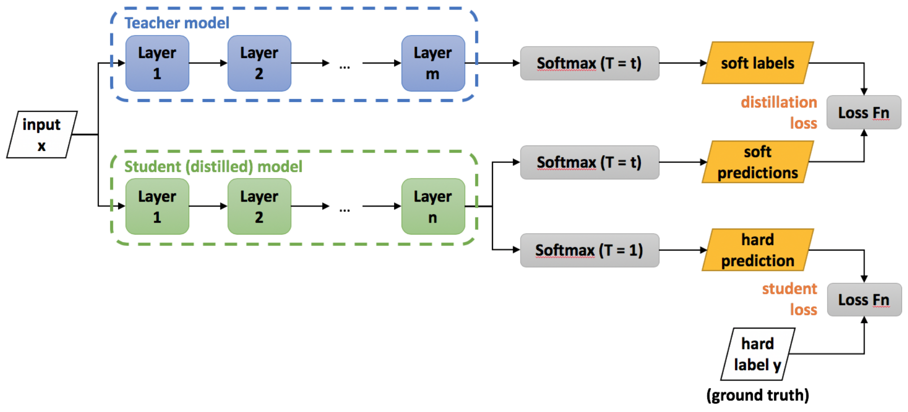
- x is the input prompt.
- W are the weights of the student model.
- y is the ground-truth completion corresponding to x.
- $\mathcal{H}$ is the cross-entropy loss function.
- z_t and z_t are the logits from the teacher and student models respectively.
- σ is the softmax function parameterized by the temperature T, calculated as:

$$\sigma(z_i; T) = \frac{\exp(\frac{z_i}{T})}{\sum_j \exp(\frac{z_j}{T})}$$

Note: Here, z_i refers to a particular index in the logit vector z.

- τ is the temperature value we are using for training the student model and is a hyperparameter.
- α and β are also hyperparameters.

This loss function is minimized to update the weights of the student model via backpropagation.



In essence, the distillation loss represents a classification task where the target is the probability distribution predicted by the teacher model. But, in many cases, this probability distribution has the correct class at a very high probability with all other other class probabilities very close to 0. Thus, it doesn't provide much information beyond the ground-truth labels already provided in the dataset.

Thus, we modify the softmax function by adding the temperature T into it. As T grows, the probability distribution generated by the softmax function become softer, providing more information as to which classes the teacher found more similar to the predicted class. In the literature, this is called the "dark knowledge" embedded in the teacher model and it is this dark knowledge that we are transferring to the student model.

In the context of LLMs, since the teacher model has already been fine-tuned on the training data, its probability distribution likely closely matches the ground-truth data and won't have much variation in tokens. By adding the temperature to the softmax, the student model receives more information in the form of a *set* of tokens that are closer to the ground-truth data (since multiple tokens will have high probabilities).

The student loss just represents the standard loss (where $T = 1$) between the student's predicted class probabilities and the ground-truth labels.

In the literature:

- $\sigma(z_t; T = \tau)$ - that is, the softer distribution produced by the teacher model for the input prompt x - is called **soft labels** (plural since it will have high probabilities in multiple places).
- $\sigma(z_s, T = \tau)$ - that is, the softer distribution produced by the student model for the input prompt x - is called a **soft predictions** (plural due to the same reason).
- $\sigma(z_s; T = 1)$ - that is, the actual prediction by the student model - is called a **hard prediction**.
- The ground-truth label y is called a **hard label**.

In the end, we have a smaller model which can be used for faster inference in a production environment.

In practice, distillation is not very effective for decoder-only models such as GPT and is typically more effective for encoder-only models such as BERT.

Post-Training Quantization

This is different from quantization during training, which is also called quantization-aware training (QAT).

Once we have trained a model (with or without quantization), we can perform post-training quantization (PTQ) to reduce the size of our LLM and optimize it for deployment.

PTQ transforms a model's weight to a lower-precision representation such as 16-bit floating point (FP16 or BFLOAT16) or 8-bit integers (INT8). This reduces the model size and memory footprint, as well as the compute resources needed for model serving.

PTQ can be applied to the model weights or both to model weights and the activations. In general, quantization approaches that include the activations can have a higher impact on model performance (performance can go down).

It also requires an extra calibration step to statistically capture the dynamic range of the original parameter values.

PTQ has a trade-off as it has an impact on model performance by sometimes leading to a small percentage reduction in evaluation metrics, but the impact can often be worth the cost savings and performance gains.

Model Pruning

In model pruning, we reduce model size by eliminating model weights with values close or equal to zero since they are not contributing much to overall model performance.

Model pruning techniques broadly fall into three categories:

- Those that require full model retraining.
- Those that are under PEFT.
- Those that focus on post-training pruning.

In theory, this reduces model size and improves performance. In practice, a very small percentage of LLM weights are zero, which nullifies the performance gains.

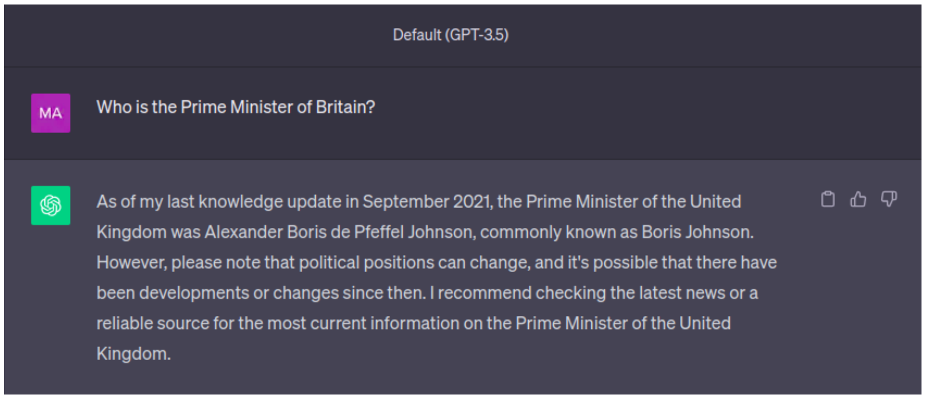
Inference Challenges in LLMs After Training

Training Cut-Off

The internal knowledge held by an LLM cut-offs at the moment of pre-training. In other words, the LLM becomes outdated due to the fact that the dataset is of finite quantity and has been collected up to a certain time period.

For example, if we trained a model in early 2022 and ask *Who is the British Prime Minister?*, it is likely to answer *Boris Johnson*. Johnson left office in late 2022 but the model has no knowledge of this since the event occurred after its training.

Below is an example from the current (September 25, 2023) version of ChatGPT (GPT 3.5). Notice the disclaimer added to the completion:



Who is the Prime Minister of Britain?

As of my last knowledge update in September 2021, the Prime Minister of the United Kingdom was Alexander Boris de Pfeffel Johnson, commonly known as Boris Johnson. However, please note that political positions can change, and it's possible that there have been developments or changes since then. I recommend checking the latest news or a reliable source for the most current information on the Prime Minister of the United Kingdom.

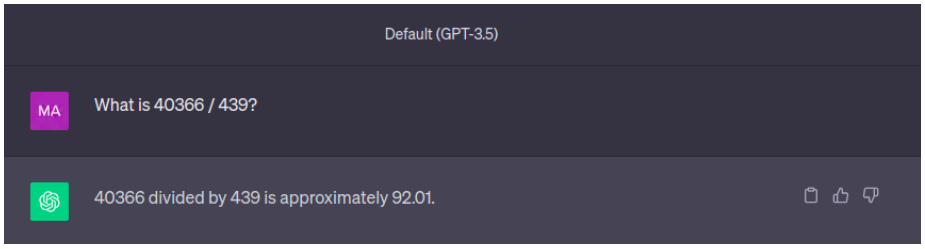
Complex Math

LLMs also tend to perform poorly on complex math problems. If we ask the model to behave like a calculator, it may get the answer wrong depending on the difficulty of the problem.

For example, if we ask the model *What is 40366 / 439?*, it may generate an answer like *92.549*. This is close to the actual answer (91.949) but still incorrect.

This is because LLMs do not actually carry out mathematical operations and are still just trying to predict the next-best token based on their training. It's not necessary that the next-best token matches the correct answer.

Below is an example from the current (September 25, 2023) version of ChatGPT (GPT 3.5).



What is 40366 / 439?

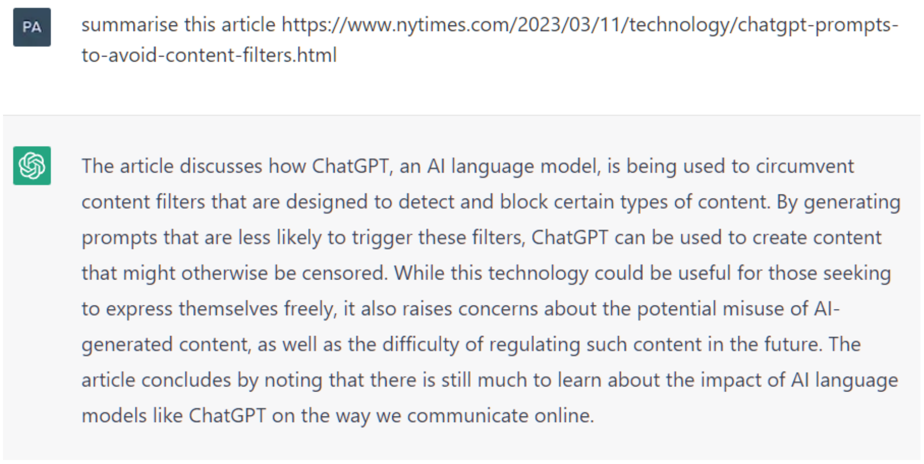
40366 divided by 439 is approximately 92.01.

Hallucination

LLMs have a tendency to generate text even when they don't know the answer to a problem, called hallucination.

For example, we can ask the LLM *What is a Martian Dunetree?* and it might respond *A Martian Dunetree is a type of extraterrestrial plant found on Mars*. Despite there being no evidence of life on Mars, the model is happy to respond with confidence.

Below is an example from an old version of ChatGPT (GPT 3.5).



summarise this article https://www.nytimes.com/2023/03/11/technology/chatgpt-prompts-to-avoid-content-filters.html

The article discusses how ChatGPT, an AI language model, is being used to circumvent content filters that are designed to detect and block certain types of content. By generating prompts that are less likely to trigger these filters, ChatGPT can be used to create content that might otherwise be censored. While this technology could be useful for those seeking to express themselves freely, it also raises concerns about the potential misuse of AI-generated content, as well as the difficulty of regulating such content in the future. The article concludes by noting that there is still much to learn about the impact of AI language models like ChatGPT on the way we communicate online.

The article does not exist and in fact, even if it did GPT 3.5 has no way of accessing the link to know what the article says. Despite this, it happily generates a summary with confidence.

Solving Inference Challenges

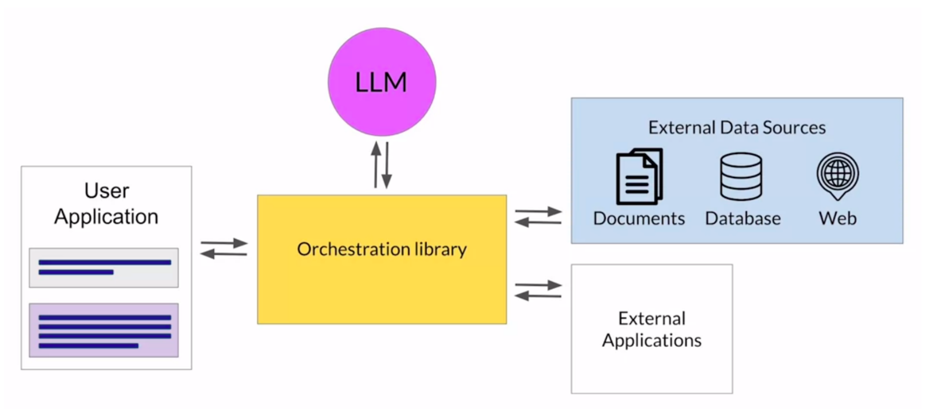
Introduction

To solve these issues, we need to connect the LLM to external data sources and applications.

We need to do a bit more work to connect an LLM to external components and fully integrate everything for deployment within our application. The application must manage the passing of user input to the LLM, as well as the return of completions from the LLM.

This is often done through some type of orchestration library.

> **Examples**: LangChain, Haystack, LlamaIndex.



This layer can enable some powerful technologies that augment and enhance the performance of the LLM at runtime by providing access to external data sources or connecting to existing APIs of other applications.

Retrieval Augmented Generation (RAG)

Retrieval Augmented Generation (RAG) is a framework for building LLM-powered systems that make use of external data sources and applications to overcome some of the limitations of these models.

Useful Resources

Reducing Model Size

- Distillation paper - Distilling the Knowledge in a Neural Network.
- PyTorch Tutorial on Distillation.
- PyTorch Tutorial on Quantization, including PTQ.
- TensorFlow Tutorial on PTQ.
- PyTorch tutorial on Model Pruning.
- Weights and Biases Tutorial on Model Pruning.

Inference Challenges

- Criticism of LangChain on HackerNews.